

TDD & Architecture hexagonale

Stratégie de tests
à travers un exemple

afup  LORRAINE



Jean-Pascal LANGINIER



Développeur PHP depuis 2011



Travail sur plusieurs CMS et Frameworks



Plusieurs années en solo dev



Calmedica

Mettre la e-santé à la portée de tous



Editeur d'un SaaS Médical



La sécurité est prioritaire sur la vitesse



Envoi de SMS selon un protocole aux patients



Analyse de leurs réponses



Le Test Driven Development



La stratégie de tests



L'architecture hexagonale



Exemple



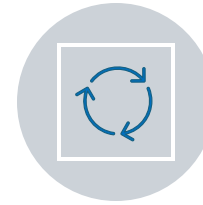
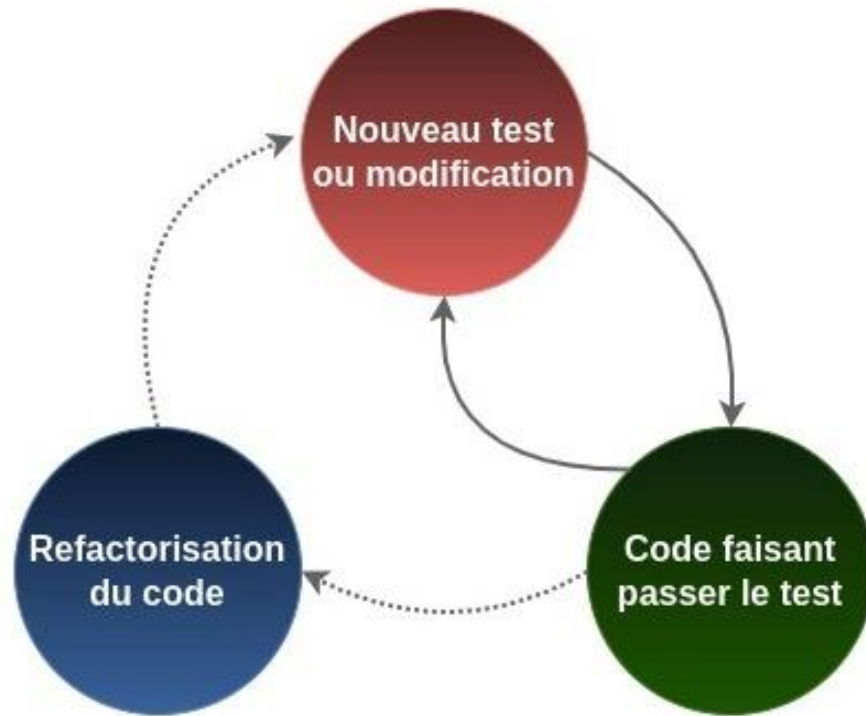
Les types de tests



Les tests de mutation

Au programme

Le Test Driven Development



PETITES ITÉRATIONS



ROUGE -> VERT :
CHEMIN LE PLUS RAPIDE



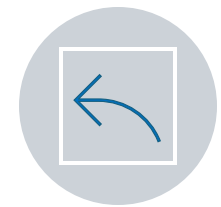
REFACTO QUAND
NÉCESSAIRE



TEST OU CODE STABLE



GARDER DE CÔTÉ LES
INTENTIONS
DÉCOUVERTES AU FIL
DU CODE



NÉCESSITÉ D'UN
FEEDBACK RAPIDE

Intérêts du TDD



Découverte du code depuis les intentions



Pas d'over-engineering



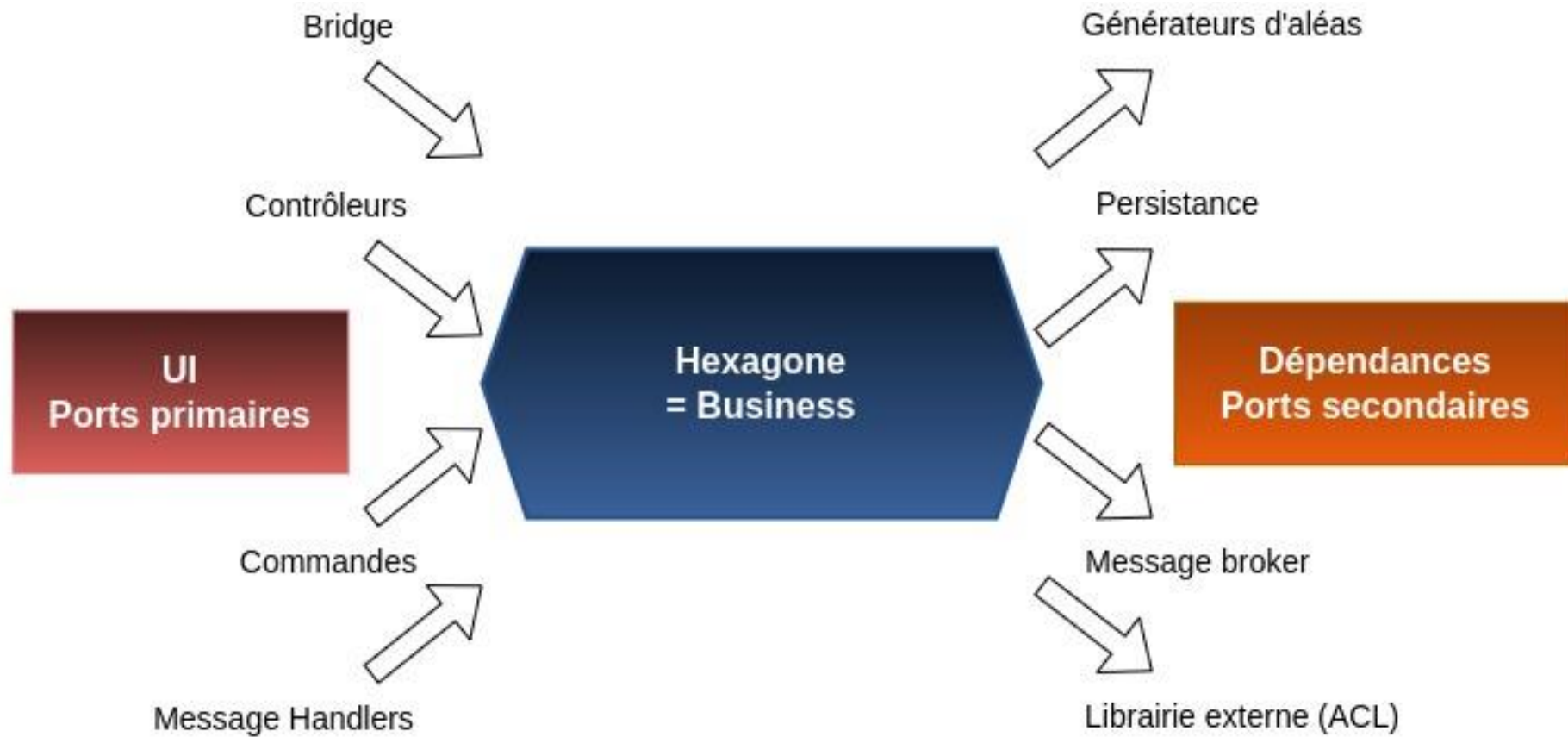
Les tests servent de documentation



La sensation d'avancer tout le temps vers l'objectif



Plus d'attachement au code



Architecture Hexagonale

Hexagone : côté UI



MyUseCase : Port primaire



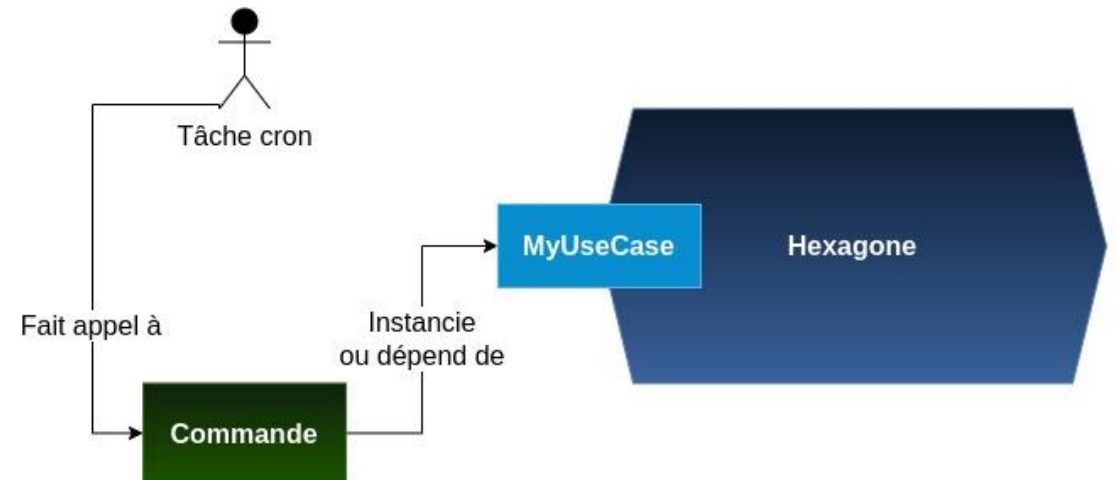
Commande : Adaptateur primaire



Possibilité de créer une MyUseCaseInterface



Le métier n'a pas de connaissance de l'appelant



Hexagonal : côté dépendances



Utilisation de l'inversion de dépendances avec des interfaces



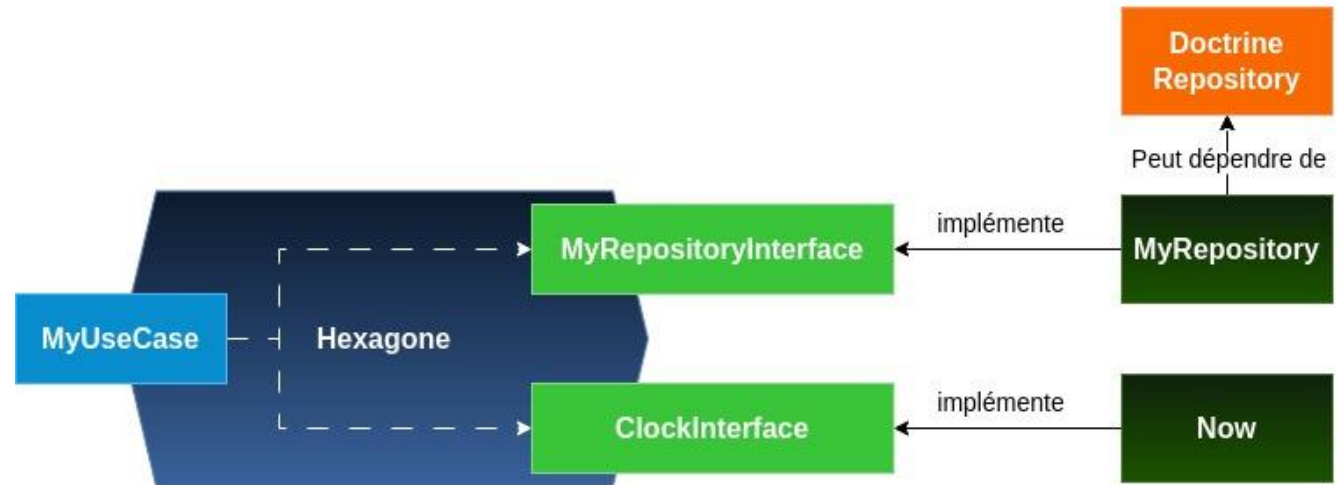
Injectées au construct des use cases

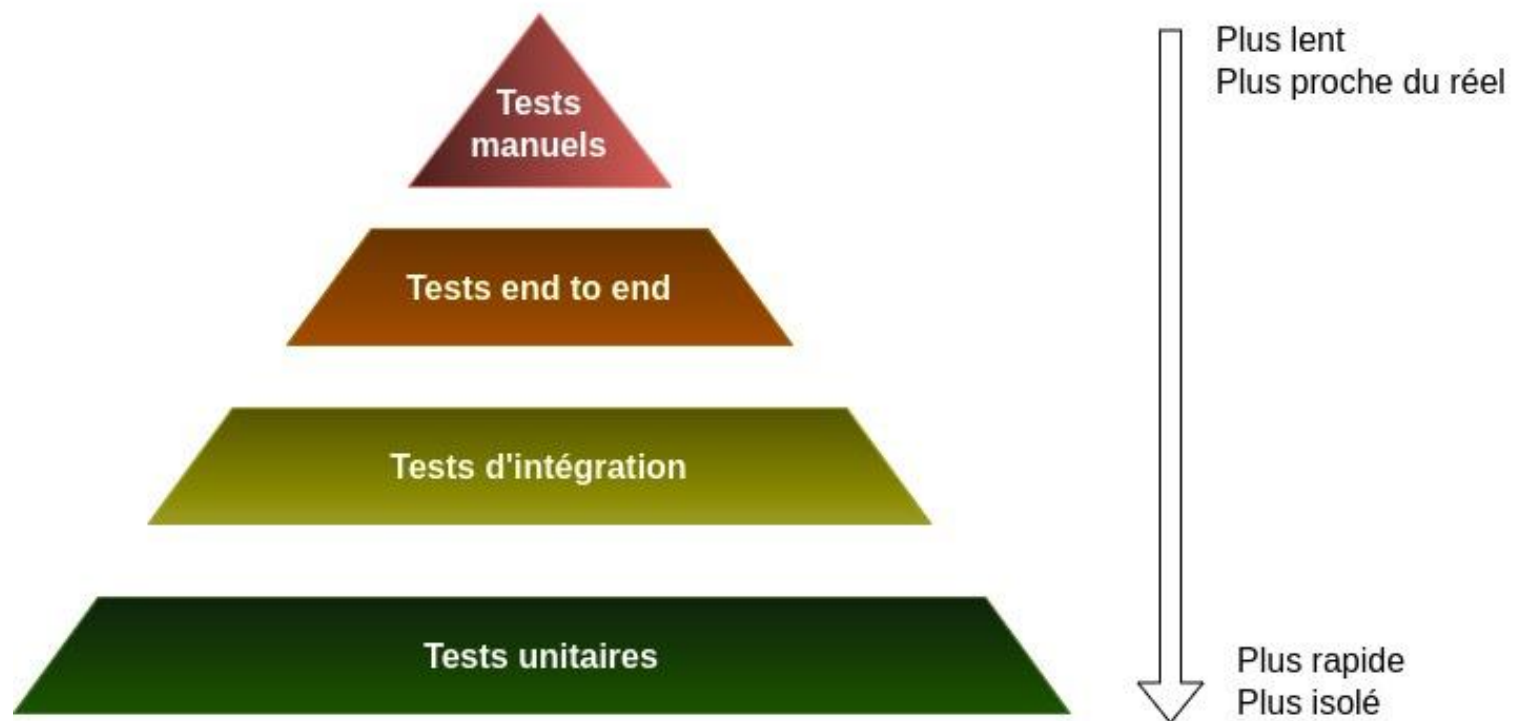


Aucun détail technique ne leak dans le métier



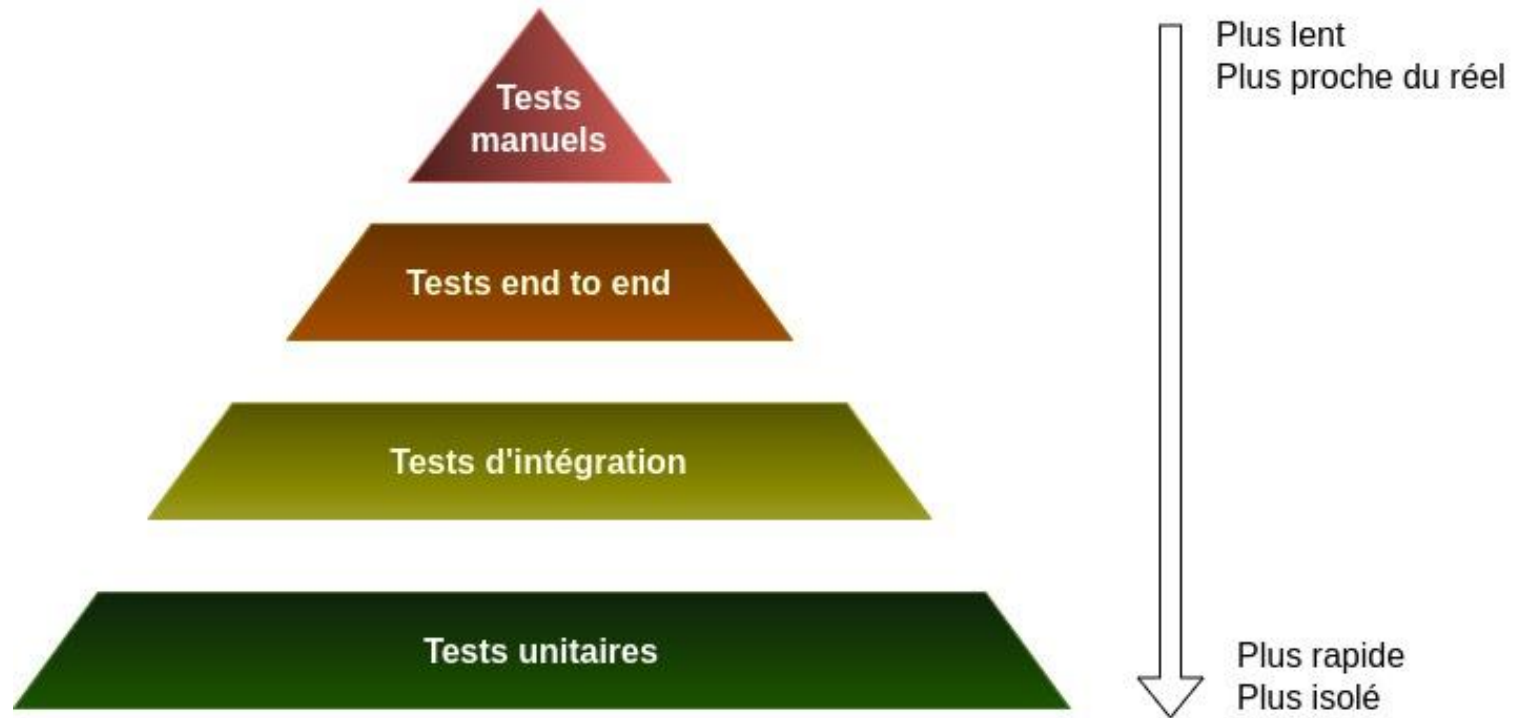
Les dépendances ont du sens niveau métier, pas au niveau "schéma"





Avant de relier les 2 : la pyramide de tests

Oui, mais ...



Avant de relier les 2 : la pyramide de tests

La pyramide ne suffit pas



Elle est globalement vraie, mais ça dépend de l'application



Elle ne pose pas la question de pourquoi on utilise tel type de test (elle n'oppose que la vitesse à la "réalité" du test)



Elle ne permet pas de définir ce qu'est chaque type de test

Type : Le test unitaire

- Test d'une unité ~~de code~~ **d'intention**
- Peut traverser de multiples classes
- Permet de tester sans charger de dépendance technique
- Donc permet de tester des centaines d'intentions en 1 seconde.

Type : Le test d'intégration

- Test du comportement d'une brique technique
- Exemple : permet de s'assurer que la connexion à une base de données fonctionne
- Etat initial à chaque début de test = pas de fixtures complètes chargées avant
- Rappel : les tests peuvent être lancés dans n'importe quel ordre

Type : Le test end to end

- Test du comportement d'un contrôleur, d'une commande, ...
- Test particulièrement lent, il en faut le moins possible
- Trop de tests end to end = la fin de l'utilisation d'une suite de tests complète comme feedback par le développeur

Mettons tout ça ensemble



Mettons tout ça ensemble



1. Tests unitaires

- Liste des intentions métier
- Développement en TDD
- Découverte du code et des dépendances

Mettons tout ça ensemble



Mettons tout ça ensemble



Un peu de code avec un exemple

- Variation du kata "Gilded Rose"
- Basée sur Symfony
- Règles métier diverses
- Utilisation d'une persistance, d'un messenger, ...
- Trouvable sur le repository :
<https://github.com/jplanginier/afup-2024-demo>
- Utilisation d'une liste de produits
- Chaque produit a un nom, une valeur et une durabilité
- Chaque jour : valeur + 1, durabilité - 1
- Multiples cas spéciaux
- Si un produit atteint une valeur de 0 : envoi de l'info au commissaire-priseur

Liste des intentions

```
class UpdateProductsAfterADayPassesTest extends TestCase {  
    public function testItWorks(): void {  
        $this->assertTrue(true);  
    }  
  
    // product has a name, a value, and a durability in days  
    // products are taken from a repository  
    // products are iterated over (not all products are loaded)  
    // standard product = 1 day passes : -1 durability, + 1 value  
    // if durability < 0 : 1 day passes : -1 durability, -1 value  
    // if is a cheese from a list, +3 value if durability is positive, -10 if negative  
    // if named "doom hammer", legendary : value is set to 1000, durability won't move  
    // if named "Ticket : xxx" : value + 5 when durability positive,  
    //     goes to 0 when durability is null (or negative)  
    // if named "Curse of xxx", stop all process and throw Exception :  
    //     the product would corrupt our shop !  
    // product with null or negative value :  
    //     message to auctioneer to remove the item later that day  
}
```

Premiers tests

```
class UpdateProductsAfterADayPassesTest extends TestCase {  
    public function testProductsAreLoadedFromRepository(): void {  
        $repository = new StaticProductRepository();  
  
        $sut = new UpdateProductsAfterADayPasses($repository);  
        $sut->__invoke();  
  
        $this->assertTrue($repository->wasCalled());  
    }  
  
    public function testProductChangedAfterADay(): void {  
        $repository = new StaticProductRepository(  
            new Product(name: 'my product', value: 15, durability: 7)  
        );  
  
        $sut = new UpdateProductsAfterADayPasses($repository);  
        $sut->__invoke();  
  
        $modifiedProduct = $repository->getByName('my product');  
        $this->assertEquals(6, $modifiedProduct->durability());  
    }  
}
```

```
class UpdateProductsAfterADayPasses {  
    public function __construct(private ProductRepositoryInterface $productRepository) {}  
  
    public function __invoke() {  
        foreach($this->productRepository->productsIterable() as $product) {  
            $this->productRepository->updateProduct($product->aDayPasses());  
        }  
    }  
}
```

Premiers tests : le use case

Premiers tests : la doublure de test

```
class StaticProductRepository implements ProductRepositoryInterface {  
    private array $products = [];  
    private bool $wasCalled = false;  
  
    public function updateProduct(Product $product): void {}  
  
    public function productsIterable(): iterable {  
        $this->wasCalled = true;  
        foreach ($this->products as $product) {  
            yield $product;  
        }  
    }  
  
    public function wasCalled(): bool {  
        return $this->wasCalled;  
    }  
}
```



```
class UpdateProductsAfterADayPassesTest extends TestCase
{
    public function testProductGainsOneValueAndLoseADurabilityWhenADayPasses(): void {}

    public function testProductLosesValueWhenDurabilityIsNegative(): void {}

    public function
testProductIsACheeseAndGetsMoreValuePerDayWhenDurabilityIsPositive(): void {}

    public function
testProductIsACheeseAndLoseALotOfValuePerDayWhenDurabilityIsNegative(): void {}

    public function
testDoomHammerIsLegendaryAndWillHaveAFixedValueAndWontLoseDurability(): void {}

    // edge case : durability is at 0 before the day passes
    // edge case : durability is at 0 after the day passes
    // ... remaining intentions
}
```

Après quelques tests

Après quelques tests : l'entité Product

```
readonly class Product{
  public function aDayPasses(): Product {
    if ($this->isDoomHammer()) {
      return new Product($this->name, 1000, $this->durability);
    }

    $durability = $this->durability - 1;
    if ($durability < 0) {
      if ($this->isCheese()) {
        $value = $this->value - 10;
      } else {
        $value = $this->value - 1;
      }
    } else {
      if ($this->isCheese()) {
        $value = $this->value + 3;
      } else {
        $value = $this->value + 1;
      }
    }
    return new Product($this->name, $value, $durability);
  }

  private function isCheese(): bool { return in_array($this->name, ['cheddar']); }
  private function isDoomHammer(): bool { return $this->name === 'Doomhammer'; }
}
```

```
abstract class ProductBase implements IdentifiedProductInterface {  
    public function __construct(protected UnidentifiedProduct $unidentifiedVersion) {}  
  
    public function aDayPasses(): IdentifiedProductInterface  
    {  
        $newVersion = new UnidentifiedProduct(  
            $this->unidentifiedVersion->name(),  
            $this->valueAfterADayPasses(),  
            $this->durabilityAfterADayPasses()  
        );  
  
        return new static($newVersion);  
    }  
  
    public function unidentify(): UnidentifiedProduct { return $this->unidentifiedVersion; }  
  
    abstract protected function valueAfterADayPasses(): int;  
    abstract protected function durabilityAfterADayPasses(): int;  
}
```

Sans toucher aux tests : refactoring

Sans toucher aux tests : refactoring

```
class Cheese extends ProductBase {  
  protected function valueAfterADayPasses(): int  
  {  
    return $this->unidentifiedVersion->durability() > 0  
      ? $this->unidentifiedVersion->value() + 3  
      : $this->unidentifiedVersion->value() - 10;  
  }  
  
  protected function durabilityAfterADayPasses(): int {  
    return $this->unidentifiedVersion->durability() - 1;  
  }  
}
```

```
class ProductIdentifier {  
    public function identify(UnidentifiedProduct $product): IdentifiedProductInterface {  
        if ($this->isCheese($product)) { return new Cheese($product); }  
  
        if ($this->isDoomhammer($product)) { return new Doomhammer($product); }  
  
        return new CommonProduct($product);  
    }  
}
```

Sans toucher aux tests : refactoring

Tests d'intégration

```
class ProductRepositoryTest extends KernelTestCase {  
    // check that a product is fetched from Doctrine DB  
    // check that multiple products are fetched, one by one  
    // edge case : there is no product in repo  
    // check insert new product  
    // check update existing product does not create a new one  
    // check updating a product won't affect another one  
}  
  
class InformAuctioneerTest extends KernelTestCase {  
    // check that message is added into messenger transport when inform is called  
}
```

Tests d'intégration : entité doctrine

```
#[ORM\Entity(repositoryClass: ProductRepository::class)]
class Product {
    #[ORM\Id]
    #[ORM\Column(length: 255)]
    private string $name;

    #[ORM\Column]
    private int $value;

    #[ORM\Column]
    private int $durability;

    public function __construct(
        string $name,
        int $value,
        int $durability
    ) {
        $this->name = $name;
        $this->value = $value;
        $this->durability = $durability;
    }
}
```

Tests d'intégration : Le repository

```
class UnidentifiedProductRepositoryTest extends KernelTestCase {  
    private EntityManagerInterface $em;  
  
    public function setUp(): void  
    {  
        $this->bootKernel();  
        $this->em = static::getContainer()->get(EntityManagerInterface::class);  
        $schemaTool = new SchemaTool($this->em);  
        $metadata = [$this->em->getMetadataFactory()->getMetadataFor(Product::class)];  
        $schemaTool->dropSchema($metadata);  
        $schemaTool->updateSchema($metadata);  
    }  
}
```



```
class UnidentifiedProductRepositoryTest extends KernelTestCase {  
    public function testProductsAreFetched(): void {  
        $sut = static::getContainer()->get(UnidentifiedProductRepository::class);  
  
        $this->em->persist(new Product('a product', 15, 15));  
        $this->em->flush();  
  
        $products = [];  
        foreach ($sut->productsIterable() as $product) {  
            $products[] = $product;  
        }  
  
        $expected = [new UnidentifiedProduct('a product', 15, 15)];  
  
        $this->assertEquals($expected, $products);  
    }  
}
```

Tests d'intégration : Le repository

Tests end to end

```
class ADayPassesCommandTest extends KernelTestCase {  
  public function testDependencyInjection(): void {  
    $application = new Application(self::$kernel);  
  
    $this->expectNotToPerformAssertions();  
    $command = $application->find('app:aDayPasses');  
    $commandTester = new CommandTester($command);  
    $commandTester->execute([]);  
  }  
}
```

Pour finir : les tests de mutation



Nécessite d'avoir toutes les intentions testées



Création de "mutants" (modification du code) et lancement de la suite de tests



Exemples de mutant

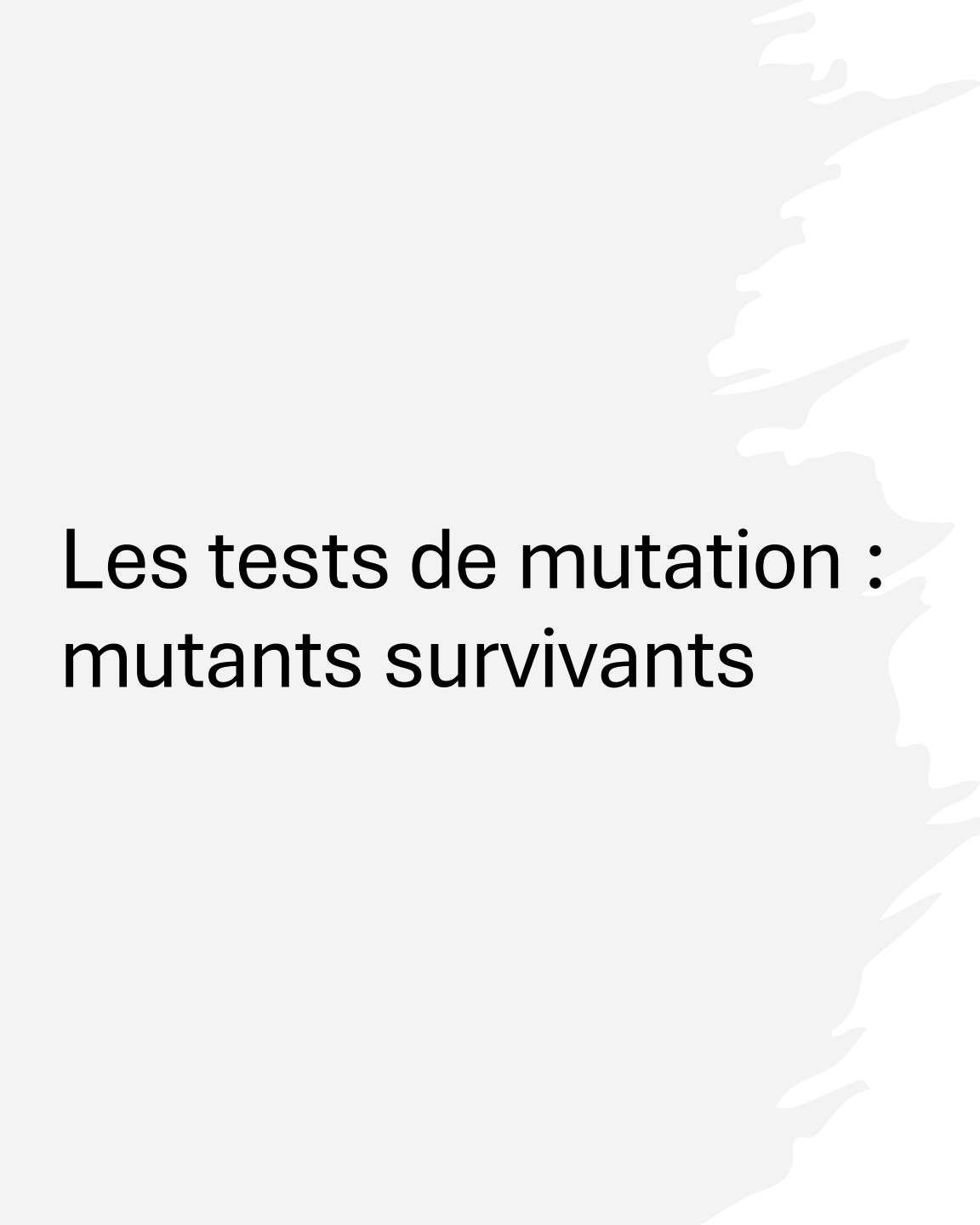
Supprimer un return

Enlever un cast

Changer une égalité en inégalité



Attention à la durée d'exécution des tests !



Les tests de mutation : mutants survivants

- Chaque mutant n'ayant fait fail aucun test est un survivant
- Raisons d'un survivant :
 - Manque d'une intention vis-à-vis du code créé
 - Complexité accidentelle
 - Faux positifs

All files Infection


















Mutants



Tests

All files

68

File / Directory		Mutation score	Killed	Survived	Timeout	No coverage	Ignored
 All files		 95.77	68	3	0	0	0
 Command/ADayPassesCommand.php		 100.00	2	0	0	0	0
 Dependencies		 100.00	3	0	0	0	0
 Entity/Product.php		 100.00	3	0	0	0	0
 Message/InformAuctioneerMessage.php		 100.00	1	0	0	0	0
 Module/GildedRose		 94.74	54	3	0	0	0
 Repository/ProductRepository.php		 100.00	5	0	0	0	0

Les tests de mutation :
sur l'exemple

Les tests de mutation : sur l'exemple

```
25         $this->productRepository->updateProduct($alteredProduct->unidentify());
26
27 -         if ($alteredProduct->value() <= 0) { ●
+         if ($alteredProduct->value() < 0) {
28             $this->informAuctioneer->inform($alteredProduct->unidentify());
29         }
30     }
31 }
32 }
```

🐞 LessThanOrEqualTo Survived (27:13) 🚩 More

☔ Covered by 8 tests (yet still survived)



Code métier testé unitairement en isolation



Dépendances techniques testées en intégrations, isolées du métier



Contrôleurs testés en E2E sur l'injection de dépendances uniquement



TDD pour assurer que les intentions sont présentes dans le code et qu'il soit "jetable"



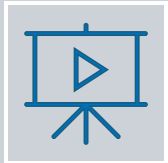
Tests de mutation pour découvrir les artefacts de code en trop et les edge cases oubliés

En résumé

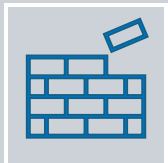
Merci de m'avoir suivi !



Retrouvez l'exemple sur
<https://github.com/jplanginier/afup-2024-demo>



Une vidéo détaillant plus longuement le
process du TDD : https://youtu.be/nbSaq_ykOI4 (Wealcome TDD)



Pour réfléchir à l'architecture que l'on peut adopter au
sein d'un hexagone, et pour définir comment diviser son
code en hexagones, regardez du côté du Domain Driven Design

openfeedback,

